Reverse Engineering Report – Crackme Challenge

Author: **Ali Rashchi** Github: **AliMMA17** Date: **2025-08-21**

Contact: **alirashchi7@gmail.com**

Legal & Ethical Considerations

This work was performed exclusively for educational purposes on a publicly available crackme challenge designed for reverse engineering practice. The patched binary will not be redistributed. All modifications and analysis respect the platform's guidelines.

Challenge information

Challenge: https://crackmes.one/crackme/688e385a8fac2855fe6fb2b5

Title: **Coder_90's keygenme90**

Platform: **Windows (x86)**

Difficulty: **2.0**

Language: **C/C++**

Objective

The objective of this challenge was to reverse the password generation logic and produce a valid keygen that generates correct passwords for given usernames.

Tools Used

- x32dbg
- CFF Explorer
- Python file (for keygen implementation)

Methodology

1. Loaded the crackme in x32dbg and monitored execution after entering username/password.

2. Searched for the Base64 alphabet string in the binary; found it in .rdata which hinted at encoding logic.

3. Traced the call at 00401978 → 4010A0, where the username is processed.

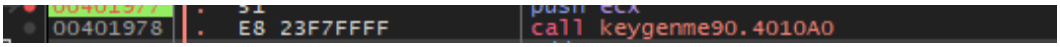
4. Observed that each byte of the username was XORed with a single constant (K).

5. Confirmed that the transformed buffer was then Base64-encoded using the alphabet found in .rdata.

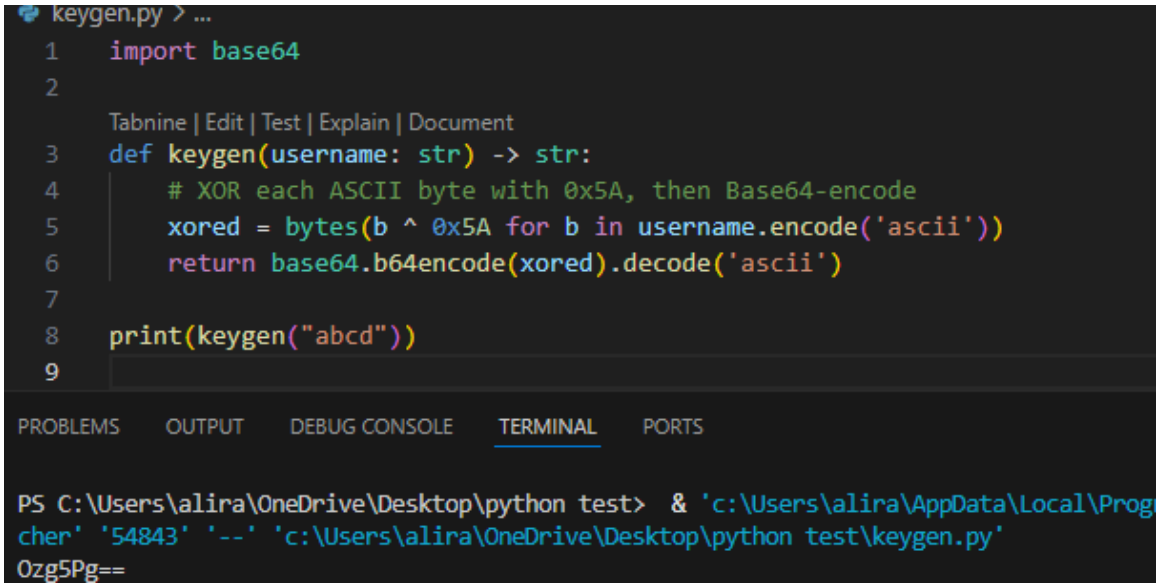
6. The resulting Base64 string was compared against the user-entered password.

Screenshots and Addresses

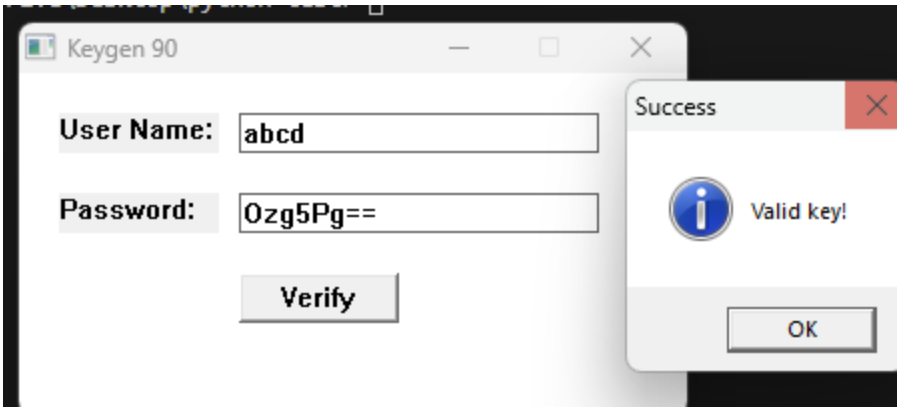
Below are the screenshots:



XOR and base64 logic in this function call



Python code



Results

The algorithm was fully recovered. The password must be generated as:

password = Base64(XOR(username, K))

Where K is a fixed single-byte constant inside the binary. A Python keygen was successfully written to automate this process.